Domain and Type Enforcement Firewalls

Karen A. Oostendorp
Lee Badger
Christopher D. Vance
Wayne G. Morrison
Michael J. Petkac
David L. Sherman
Daniel F. Sterne
Trusted Information Systems, Inc.
3060 Washington Road Glenwood, Maryland 21738

Abstract

Internet-connected organizations often employ an Internet firewall to mitigate risks of system penetration, data theft, data destruction, and other security breaches. Conventional Internet firewalls, however, impose an overly simple inside-vs-outside model of security that is incompatible with many business practices that require extending limited trust *to external entities, for example, suppliers, bankers, accountants, advisors, consultants, partners, customers, and allies. Additionally, firewall security perimeters are somewhat weak: they provide no protection from inside attacks and do not protect sensitive data, which can be exported by tunneling through permitted protocols. As the Internet evolves towards applets, mobile agents, and object frameworks, these problems likely will worsen.*

This paper reports on our experience with an enhanced security firewall based on Domain and Type Enforcement (DTE), a strong but flexible form of access control. A DTE firewall provides several benefits. First, it runs application-level proxies in restrictive domains, thereby increasing security, and runs network services such as HTTP and FTP under DTE controls, thereby reducing risks that network-based attacks will compromise local resources. Second, a DTE firewall coordinates role-based security policies that span networks by passing DTE security attributes between DTE clients and servers. These attributes allow security policies at the endpoints to be coordinated; such coordination adds defense in depth *to the traditional firewall security perimeter: this permits safe exportation of normally risky services such as NFS. Finally, a DTE firewall interoperates with "non-DTE" systems and associates DTE security attributes with these systems so their interaction with DTE clients or servers can be controlled. We describe here the design of a prototype DTE firewall system and informally evaluate its security, compatibility, functionality, and performance.*[1]

1. Introduction

Internet-connected organizations often employ an Internet firewall to mitigate risks of system penetration, data theft, data destruction, and other security breaches. Conventional Internet firewalls, however, impose an overly simple inside-vs-outside model of security that is incompatible with many business practices that require extending *limited trust* to external entities, for example, suppliers, bankers, accountants, advisors, consultants, partners, customers, and allies. Additionally, firewall security perimeters are somewhat weak: they provide no protection from inside attacks and do not protect sensitive data, which can be exported by tunneling through permitted protocols. These problems likely will worsen in the next few years as applets [9], mobile agents [19], and object frameworks [12] provide opportunities for malicious programs to execute behind firewall security perimeters.

While firewall security perimeters are helpful, we believe that a fundamentally stronger mechanism will be required to protect "sacrificial lamb" network servers, to run network clients in environments that control damage inflicted by malicious servers (and applets),

[1] This research was supported by ARPA contract DABT63-95-C-0018.

2 Background

The real-time components of Triad represent the evolution of real-time systems that span the domain of complex, heterogeneous, industrial and military systems. In early real-time systems the design focus was sheer speed of execution, using minimal executives and careful application tuning, in order to synchronize with external world events. As processor power increased and costs dropped, more generalized approaches for designing and re-using real-time system software began to emerge. Modern real-time system designs take advantage of the techniques of computational predictability analysis and adaptable scheduling algorithms. The basic intent is to enable the completion of a function *at the right time* taking into account computation time, resource access time, and other factors, rather than simply striving to perform the function with as few instructions as possible.

This change of focus from *fast* to *predictable* allows a richer functional mix within a single real-time system. For example, it enables the incorporation of distribution technology which, in turn, enables the use of redundancy and replication of processing resources to increase a real-time system's survivability.

The need for multi-level security is becoming manifest in parallel with the increasing technical scope of real-time systems. The drive to integrate multiple applications brings with it the need to separate the applications algorithms and data from one another since they may embody drastically differing sensitivity levels and integrity levels. The traditional "really fast" real-time design pitted the use of trust mechanisms against mission success. However, the modern generalized real-time system design avoids that pitfall and largely mitigates many such design goal conflicts. In addition, the use of "system build" techniques [4] (described in Section 4.1.1) is used to further eliminate any remaining potential conflicts that can arise at run-time.

Distributed system requirements are an additional factor of the increasing scope of both real-time systems and MLS systems. A distributed system is a collection of computers connected by a high-speed interconnect or network. This collection, or cluster, is controlled by software that makes the underlying interconnect largely invisible. As a result, ordinary users and application software view the cluster as a single computer system, and can remain unaware of its distributed nature. Such distribution functionality is essential not only for geographically distributed command and control software, but also for real-time systems which require resource redundancy for fault tolerance.

The Triad Project is a research effort sponsored by Rome Labs, awarded August 1993, to produce a robust proof-of-concept prototype demonstration in December 1995. We will complete requirements definition stage in mid-1994 and the software detailed design in the second half of the year. In addition to the technical goals described here, the Triad project will employ an integrated 2167a development process which incorporates TCSEC security documentation and review with that of generic software development. The project stems from earlier Rome Lab sponsored research in the area of multi-level secure distributed operating systems (MLS DOS study) [2].

3 Approach

The Triad System will incorporate modern operating system features from the three basic areas, to form a TMach-based operating system supporting the basic functionality identified by the MLS DOS study effort, specifically its programming abstraction, scheduling algorithm, and formal security policy approach. The use of the programming abstraction (threads making distributed object-oriented remote procedure calls) is described in Section 5, and is facilitated by the addition to TMach of shuttles, described in Section 5.3. The scheduling algorithm usage is in the context of the replaceable scheduler interface work, described in Section 5.2. Finally, the Triad policy has already been formulated by merging elements of the MLS DOS policy and the TMach policy.

Various features of TMach are central to our approach. Most importantly, TMach provides the MLS functionality needed for Triad. Our decision to use TMach is based in part on the judgement that it is easier to add real-time and distributed processing functionality to a MLS system than it is to add security to a real-time and/or distributed system. Due to TMach's B3 layered architecture and modular implementation, it turns out that real-time and distribution extensions are required for a small number of components, and that the real-time extensions are largely orthogonal to the distribution extensions.

Likewise, since TMach is designed to be extensible, it lends itself well to the kind of extensions needed for the Triad system and Triad applications. Being composed of several distinct servers, TMach can be readily augmented by new servers, such as application-level servers implementing new application types of objects. The object-oriented design of TMach server also means that they can be readily augmented with new subsystems to implement distribution. Further extensibility is derived from the Mach microkernel [5] which was designed to be extended into a distributed environment in the manner required for Triad. Thus, not only does extensibility at both the TMach server level and Mach microkernel level support the addition of distribution functionality, but this extensibility was explicitly designed partly for distribution.

Finally, because TMach is built on top of the Mach microkernel, it has portability benefits that are critical for distribution. The microkernel is structured so that all hardware-specific (non-portable) software is isolated, so as to permit straightforward modification for porting to multiple platforms. The TMach software itself has no hardware-specific portions. As a result, it can run on any hardware base to which a suitable version of the microkernel has been (or will be) ported. This portability is essential to future use of Triad, in that some distributed applications may require operation in an environment composed of heterogeneous hardware bases.

4 Trade-off Analysis

In addition to the MLS DOS study foundation and the TMach basis, trade-off analysis is a key part of the Triad design approach. Each of the three areas of requirements has ramifications for the other two, and we consider these interactions in terms of three pairs of one area against another, each of which is described in the following sections.

4.1 Real-time Processing versus Security Requirements

In designing the Triad System to provide both real-time processing and satisfy security requirements, several trade-offs must be assessed. There are several areas where security versus real time appear to be in direct conflict, and then there are other areas where the trade-offs required are less dramatic. The following issues in interactions between security and real time are addressed in the Triad design:

Scheduling in any sophisticated scheme provides the potential for covert channels in a trusted system. In some sense, covert channels due to scheduling algorithms are unavoidable. However, there is in fact a very much reduced threat of covert channel exploitation because of certain environmental characteristics of many real-time systems: systems which provide real-time processing are intended for use in a closed processing environment, rather than a general development-oriented OS environment. Such closed environments typically do not support a general programming environment and have extremely limited or no human interaction during real-time processing. These aspects of trusted real-time systems greatly reduce the threat of covert channel exploitation. The Triad approach is to address these issues through a design that satisfies real-time scheduling goals while reducing the opportunity for modulation of covert channels.

Responsiveness versus potential covert channel bandwidth is an issue because real-time systems are designed to be highly responsive to application needs, i.e., provide rapid predictable service times. This is particularly true in the case of the Benefits Accrual (BA) scheduling model which forms the basis of scheduling in the Triad System. On the other hand, this trade-off is potentially challenging because in the secure systems community, the traditional approach to reducing or eliminating covert channels is to add constraints to the scheduler (including introducing non-determinism, partitioning resource usage by security class, and/or using static scheduling). However, the inverse properties (predictability, flexible resource usage, and dynamic scheduling) are all critical to meeting real-time requirements, so the Triad System must be designed to provide a high degree of responsiveness. Therefore, the Triad design identifies potential covert channels, and uses this information to attempt to determine the difficulty of closing or reducing the bandwidth of certain channels.

Flexibility The Triad System design includes a replaceable scheduler interface. The Triad prototype includes a scheduler module using this interface, with appropriate security assurances made for that particular scheduling module, because all scheduling must take place in the Trusted Computing Base (TCB). As a result, any application-supplied scheduler extensions would need to be accompanied by assurance evidence. Specific recommendations will be provided as to what form such assurance evidence should take.

Separation of Mediation from Enforcement TIS research has studied the trade-offs involved in the performance overhead for certain security related operations versus efficient and predictable real-time processing. We have shown that traditional access mediation can be divided into two phases—mediation and enforcement—and this concept plays a central role in the design of TMach security services. Furthermore, we have found that for many real-time systems a considerable amount of access mediation can be done a priori. As a result, mediation can be done as part of the system build process, and enforcement can be done during real-time processing.

Assurance The B3 system architecture calls for a layered, modular TCB of minimal size which uses the techniques of data abstraction and data hiding. The key to satisfaction of this requirement in the Triad system is to minimize changes to the TMach TCB. The modifications Triad will require in the kernel and servers will not affect the existing TMach layering scheme. The addition of the distributed IPC layer will not violate good layering principles, since it will offer services to the TCB layer but not require services of the TCB layer; in turn, the distribution layer will use services offered by the kernel, but not offer any services relied upon by the kernel.

4.1.1 System Build

Central to each of these areas is the use of system-build techniques. Build-time options can be used to control the way that covert channels are addressed, to allow build-specific approaches to providing the kind of scheduling and responsiveness needed for that build, depending on the threats that covert channels pose in the system's actual deployed operational environment. Build-time options can also be used to specify the use of an additional or alternative scheduling module. Also of critical real-time importance is build-time access mediation (described in [4]). The system build process will assist in changing the user-oriented aspects of TMach, e.g. user login and subject creation, to an embedded system orientation which allows security subjects, objects, and their access sets to be pre-defined at system build time.

In addition, the system build process can be used to implement decisions about distribution and replication of some trusted servers. For example, some servers, such as those for Audit and Identification and Authentication (I&A) have databases that need not be replicated in cases where I&A data is static, or where changes are rare and can be handled administratively. In these cases, system build techniques can ensure consistency among copies of the database, without the use of additional run-time functionality.

4.2 Security Requirements versus Distribution

In discussing trade-offs between security requirements and distributed processing requirements, covert channels are, once again, a key issue. Distribution can potentially both exacerbate and alleviate covert channels.

In the Triad System, the most obvious way in which distribution mechanisms introduce covert channels is through the synchronization mechanisms used for coordination of distributed servers on different nodes. These synchronization mechanisms use limited resources that can be exhausted. This can produce a storage-like covert channel in a trusted multi-level server.

Another way in which distribution can exacerbate covert channels is in the area of timing channels. First, accessing resources on a remote node can introduce noticeable delays and therefore introduce possible timing channels. Secondly, in a non-distributed (single node) trusted system, the actual exploitation of timing channels can be difficult because of the noise introduced by many subjects sharing the resources. However, in a distributed system, it is possible that a set of cooperating Trojan Horses on multiple nodes could focus on a covert channel on a single node and exploit that channel more effectively than would be possible on that node alone.

However, just as there is more processing power in a distributed system which can introduce new covert channels, there are also more mitigating factors. First, there are more total resources of the combined system. These resources, if used properly, can make storage or timing channels that result from resource exhaustion harder to exploit because it will be harder to exhaust the resource. In order for this to be true, resources must be managed globally rather than locally or partitioned. Likewise, there is also more noise, from the latency of communications between nodes, which makes more difficult the exploitation of timing channels.

The common feature among these covert channel considerations is that the factors (total resources, resource allocation, communications mechanisms, communication latency) are specific to a particular operational environment. Therefore, the appropriate measures can only be completely determined at system-build time.

4.3 Real Time Requirements versus Distribution

When considering the trade-offs between real-time requirements and distributed system functionality, difficulty arises because some distributed system operations can take an arbitrary amount of time while real-time operations can have deadlines which must be met.

As a result, the design of the Triad System includes a method of ensuring real-time requirements while preserving distributed system functionality. Specifically, the Triad System provides mechanisms to trade off satisfaction of deadlines versus replication. The Triad System uses a replication protocol based on the Cronus [6] replication protocol. The important characteristic of this protocol which is particularly useful for the Triad system is the ability to vary the amount of computation needed for replication consistency. This approach will allow for considerable flexibility in making trade-offs.

In general, the approach to implementing distribution functionality is to implement it outside the execution paths of potential real-time processing whenever possible (e.g. background processing for consistency updates), and when not possible, to provide features (such as that described above) that allow tuning of the amount of distribution processing that can delay time-bound computation.

5 Triad System Architecture and Features

Triad provides object-oriented services in client-server environment, implemented in a layered architecture. This section describes the architecture and the various real-time and distribution features that are mentioned in the description of the various architectural layers.

5.1 Architecture

The Triad System has a layered architecture consisting of 5 layers:

Kernel Layer: a Mach microkernel, executing in hardware-protected space, and providing the basic abstractions to the rest of the TCB and to untrusted applications. These abstractions include the thread (the schedulable unit of execution), the task (a group of threads sharing an address space and an identity), and message-based IPC (inter-process communication) using protected capabilities called ports. This IPC service is commonly used for local RPC (remote procedure call) between threads of different tasks, and includes the shuttle mechanism.

Distribution Layer: a TCB layer which transparently extends the microkernel's IPC service to operate between hosts, providing the same IPC between threads whether or not their tasks are on the same host. This service is optimized for the cases when it is being used for remote RPC, and includes functionality for scheduling coherence.

System Security Layer: a TCB layer consisting of trusted system servers which provide access to all system resources via access-controlled objects, in accordance with the Triad security policy. Each object has a type, and each type has a specific set of operations (or methods) implemented by a manager for that type. Each manager is a trusted server. As a result, system services are implemented in a client-server, object-oriented manner.

Application TCB Layer: an application TCB layer consisting of trusted servers providing non-system services by managing application-specific types of objects needed by specific applications.

Untrusted Code Layer: an application and OS layer containing untrusted code. Application code may include untrusted clients and type managers, as well as OS servers that implement the services of a specific OS (such as Unix or DOS) on top of TMach's services.

Of these, the first three layers are part of the Triad System; the fourth and fifth layers are application specific. However, a Unix server in the Untrusted Code Layer will be available for application software which uses the Unix application programming interface (API). The affected components of the base TMach system are: the kernel, by the separation of its scheduling aspects into a replaceable scheduling module, and by the addition of the migrating shuttle mechanism for RPC; and two of the servers, where the changes will in no way affect the TMach's modularity or layering. In addition, a new server will be added to handle distributed IPC (dIPC).

This architecture enables Triad's basic concept of operation, the *invocation*. To use some system or application service, a client invokes a method on an object, resulting in an RPC to a server. The shuttle mechanism ensures the scheduling coherence of the RPC. The distributed IPC service handles cases where the server is located on a different host (or where the server invokes an operation of another server that is a different host) by transparently forwarding the invocation to the other host and handling the scheduling coherence on the other host. The following sections describe this overall functionality in more detail.

5.2 Scheduling

The real-time features of Triad are largely implemented as Kernel Layer functionality of scheduling and thread management. Real-time scheduling is provided by an implementation of Benefits Accrual real-time scheduling algorithm [7]. However, rather than replacing the Mach microkernel's scheduler with a BA scheduler, the Triad approach is to utilize current Mach research in replaceable schedulers [8].

In this approach, the internal structure of the Mach microkernel is altered so that the scheduler implementation is separated from the rest of the microkernel code in a highly modular manner. The resulting interface between the scheduler and the rest of the microkernel is referred to as the replaceable scheduler interface. This interface is general enough to accommodate the requirements of various scheduling algorithms, including BA. Within the scheduling subsystem that implements this interface there is a module that implements a particular scheduling algorithm. This module can be replaced to implement the desired algorithm for a particular system.

The Triad prototype will use a Mach microkernel with a replaceable scheduler interface and replace the standard scheduling algorithm module with a new module that implements the BA algorithm. Thus, even though the scheduling algorithm is different, the rest of the microkernel need not be changed. This is a particularly important result for the Triad System, since different real-time systems have different scheduling algorithm requirements.

Thread attributes are the focus of the thread management aspect of Kernel Layer real-time functionality. Each thread has a number of attributes, some of which pertain to scheduling (e.g. priority in the standard Mach scheduler). As part of the replaceable scheduler interface work, thread scheduling attributes are extended to support a range of scheduling algorithms. In the Triad prototype, which uses the BA algorithm, the thread scheduling attributes which will be used are the those pertinent to BA, including statistical measures of thread activity, as well as hard and soft deadlines.

5.3 Scheduling Coherence

Other than the BA scheduler, the principal Kernel Level real-time functionality in Triad is a mechanism called scheduling coherence. Scheduling coherence is a means to ensure that all the work of a particular real-time computation is scheduled in the same way. This is important when the computation may include an RPC. Consider the common client-server model, where a client makes an RPC, and the RPC is carried out by the server. When a client thread is performing some time-bound computation, the thread's scheduling attributes have been set to particular values so that the computation is likely to complete in time. While a server thread executes the client's RPC, the client thread is sleeping. If the server thread has different scheduling attributes than the client, then the server thread will be scheduled differently than the client thread, perhaps getting a smaller share of processing resources, with the result that the computation may not complete in time.

5.3.1 Shuttles for real-time RPC

We will make use of a recently developed mechanism called shuttle migration [9][10] to provide scheduling coherence for RPC. In this paradigm, the *thread* abstraction is the schedulable entity, comprising the stack and processor state. A *shuttle* comprises the scheduling policy and parameters and resource attributes. During RPC the shuttle migrates to the new task, bringing with it to the new task exactly that information required for scheduling coherence. In the server task a waiting *empty thread* attaches to the migrating shuttle and, now an *active thread*, it executes the appropriate server code to handle the message.

5.3.2 Process Isolation

We are currently analyzing the use of shuttle migration and its effect on TMach's trusted servers. One immediate area of concern is the effect on B3 process isolation and subject definition.

The task is the "process"-like abstraction of the Mach kernel, and the unit of subject definition in TMach, because it is a protected domain of execution—address space, and set of memory object and ports—that is permanently bound to a set security attribute. We need to assure that, though shuttles migrate between tasks, the usual distinction between tasks, and hence between subjects and between isolated processes, is preserved.

When a thread is executing in a task, it has access to everything in the task's address space, and it has some execution context within that address space. When its shuttle migrates to a new task on RPC, that execution state is preserved and left behind, remaining inactive until the shuttle returns to the task. The new task in the RPC chain must have an empty thread for the incoming shuttle to be bound to. When a shuttle enters an empty thread, it carries no state from the previous task other than the thread's scheduling attributes. While the shuttle is executing in the new task, the new active thread has no special access to anything in earlier tasks in the RPC chain. The same is true when a shuttle exits a thread to return to the previous task. In addition, once a shuttle has exited a thread, the exited thread retains no state information.

The security-critical user identity is maintained as well—the user identity associated with a thread is the user identity of the task in which the thread is running. The information flow between the calling task and the called task of a migrating shuttle RPC is restricted to the thread's attributes, the calling task's RPC message, and called task's reply message; and these two messages are exactly the same information that is passed in a traditional RPC.

As a result of all these properties of shuttle migration, all the security-related attributes of the traditional thread model remain with the task-specific thread. Therefore, the critical "process isolation" security requirement is still met, and the definition of the subject is unchanged.

5.4 Distributed IPC

The microkernel provides an IPC service which is the basis of all communication between tasks on the same host. The distributed IPC service is exactly the same service as the microkernel's IPC, but provides for communication between tasks that are on different hosts in a distributed system. The Triad System component that implements this service will be based on the x-kernel implementation of Mach IPC between networked hosts [11].

5.4.1 Distributed IPC Server

The distributed IPC mechanism is implemented by the distributed IPC server. With local IPC, one task sends a message to another using a port held by the other task. The port is a protected capability representing a message queue which is held by one receiving task, and to which messages can be appended by potentially multiple sending tasks. When one task sends a message to another task which is on a different host, the dIPC server is involved as an intermediary. The sending task does the message-send over a port, but the receiver of that port is not really some task on another host; this is not possible because the kernel only knows about tasks on the host it manages. Instead, the receiver of the port is the dIPC server, which receives the message from the sender. This port represents a port on another host, where the message's real destination task runs. The transmission of the message between these two client tasks is the result of the cooperation between the dIPC server on the sender's host and the dIPC server on the receiver's host. In other words, the dIPC server acts as a stand-in for remote tasks that local tasks can communicate with; and local ports received by the dIPC server are stand-ins for ports that those remote tasks are receivers for.

When the dIPC server receives a message over a local port, it checks to see what remote host and remote port correspond to the local port. Then, the message data (and other information including which port it is bound for) is sent over the network to the dIPC server on the receiver's host. This dIPC server sends the message via local IPC on the port received by the actual receiver of the message.

The dIPC server shares critical security-relevant functionality with the Mach microkernel: both are responsible for propagating identification data which is essential for the TMach TCB to enforce access controls. Each TMach task has an attribute called a security identifier (securityID) which represents the user, groups, etc. of the human associated with the task. One important feature of the Mach microkernel's

IPC service is that each message is tagged with the securityID of the sending task. This sender securityID is critical to the enforcement of the security policy by the TMach TCB. The dIPC server transmits the sender securityID with the message data over the network to the remote dIPC server, which sends the original securityID in the message to the receiver of the message. Since the dIPC is separate from the kernel, no kernel modifications will be required, nor will there be an affect on kernel assurance.

5.4.2 Distributed Scheduling Coherence

Another function of the distributed IPC service is implementing scheduling coherence by providing distributed emulation of local thread migration. This emulation is needed because Mach microkernel threads are inherently local abstractions (the Mach microkernel is unaware of other hosts). Hence, a Mach microkernel thread cannot truly migrate to another host. However, the dIPC server can implement a close analog of thread migration.

When the dIPC server is handling an IPC message that is the outgoing part of a migrating thread RPC, some specific actions are needed to maintain scheduling coherence. With each message's data that is transmitted to another host, the dIPC server must also send some more information about the thread that sent the message: its scheduling attributes. This information is used by the receiving dIPC server to perform its distributed emulation of thread migration.

Once a thread migrates into the dIPC server and the outgoing message data is transmitted, the thread goes to sleep. On the receiving machine, the dIPC server uses a local thread to act for the sending thread on the other machine. This thread migrates from the dIPC server to the message's destination task. But before doing so, the dIPC server ensures that this thread has the same scheduling attributes as those that arrived with the message, i.e. the scheduling attributes of the sending thread. As a result, the receiving host's RPC processing is scheduled coherently with the computation in the sending thread on the other host.

5.5 Servers and Replication

Although the basic real-time and distribution functionality is implemented in the kernel and distribution layers of Triad, upper layers also have a role to play.

With respect to real-time, there are four higher-level aspects of real-time functionality. First, for those system servers that can be used by application servers or clients with real-time requirements, the system server implementation must ensure that critical functions are carefully coded so as to predictably execute in bounded time for a time-bound thread. Second, these system servers must use activations, so that time-bound client threads can migrate into the server task to execute server code (see Section 5.3.1). Third, application servers must also use activations when appropriate for the same reasons. Fourth, in order to ensure that each real-time RPC is scheduled coherently, the migrating-thread RPC interface must be used by any application servers and clients which make RPCs and which have real-time requirements.

Thus, higher-level real-time functionality is largely a matter of correctly using lower-level functionality, and this requirement is pervasive for any real-time-relevant code. For distribution, however, the situation is somewhat different. The main lower-level distribution functionality is distributed IPC, and no particular efforts are required of higher-level code to use this transparently implemented service. However, there is more to distributed service than IPC, and it is the TMach system servers that must implement the rest of the distribution functionality.

Of the TMach servers, though, there are only two servers that must provide distributed functionality. These are the Root Name Server (RNS) and the File Server (FS). The RNS is TMach's central security server which implements the reference validation mechanism (RVM) by providing access to all objects and implementing the access controls on them. In addition, the RNS implements some system object types including the directory. The File Server implements another primary system object type, the file. The remainder of the system servers either provide services whose distribution functionality can be handled by system build techniques (see Section 4.1.1), or provide services that are inherently local (e.g. device access) and which have no distributed aspect.

In Triad both the RNS and FS are replicated servers. That is, multiple hosts run an instance of the server, and each server instance cooperates with instances on other hosts, to provide a distributed service. In addition, application-level servers may or may not be replicated, depending on application requirements. If server replication is needed, then the application server implementation can be based on the same server framework as the system servers, and use the same replication library that will be used to implement Triad extensions to the TMach RNS and FS. As a result of such server replication, the service would be provided by multiple instances of the server. This increases the reliability of the service, and its availability in the face of host failures. Availability and reliability may also be preserved in additional failure modes (such as network partition), depending on the network topology, the amount of server replication, and the degree of cooperation between peer server instances.

5.5.1 Object Replication

The primary distribution functionality of the Triad RNS and FS is object replication. In addition, the RNS uses object replication to provide a global name space. The Triad RNS on each host implements the local TMach name space of that host, and extends this name space into the distributed system, via its cooperation with other RNS instances on other hosts. As a result, the name space is uniform throughout the distributed system, so that every object can be accessed by the same name regardless of which host the accessor resides on.

Object replication is technique used by replicated servers that implement a type, i.e. a set of similar objects and a set of operations on those objects. This technique allows for various degrees of availability of objects, in spite of node or server failures. Without object replication, a server that implements a particular object stores the data of that object, and performs the operations on the object using that object data. The object is globally available throughout the distributed system, since clients anywhere in the system can contact the server for service on the object—subject of course to access controls. However, if that server or its host goes down, then the object is unavailable. Even though the service itself may still be available via other replicated server instances (which manage other objects of the same type), there is no availability of objects that are managed solely by the down server instance.

This sort of single-point-of-failure availability problem can be mitigated by the use of object replication, in which multiple server instances maintain a replica of the object data. As a result, any one of these servers can provide service for the replicated object. Not every object of a type need be replicated, and not every server managing the type has to keep a replica. This, replication can be flexibly used to provide reliable service for an object in potentially several failure modes, depending on the number of replicas, their distribution within the network topology, and the nature of the object's consistency requirements,

The consistency of the object data is the key issue in object replication. By allowing multiple servers to provide service to one object, one allows the possibility that multiple servers could modify their replica's data, causing it to become inconsistent with the replicas of other servers. There are a variety of different consistency mechanisms that can be applied to this situation, but no one of them is suitable for all the different kinds of types of objects that could be replicated—or indeed even for all the objects of one type.

Therefore, the Triad approach is to implement a consistency mechanism that is flexible enough to meet various needs. To do so, Triad has adopted the version-voting mechanism of Cronus and the approach described in [6]. Within this approach, an update to a replica is predicated upon the updating server obtaining locks on other replicas from other servers. The number and/or proportion of all replicas that must be locked is a value that can be specified differently for different replicated objects. Likewise, there are settable parameters for the propagation of new values to replicas that did not participate in an update. As a result of this flexibility, Triad will implement a replication mechanism that is scalable to distributed systems of various sizes. The Triad implementation is in $C^{+}+$, an object-oriented language, in a highly layered and modular fashion. The changes required for replication will be orthogonal to the principal functionality of the servers, and can be accomplished without changing the design or structure of the servers.

6 Conclusion

The Triad project is developing a prototype Triad system designed to provide processing capabilities for real-time distributed military C^3I applications which process information of different classifications. Triad uses a base Mach microkernel and TMach trusted servers, for the required multi-level security features. This base is extended with real-time and distribution features which are used to provide services—including migrating thread RPC, distributed IPC, distributed name and file service—which clients and application servers utilize to perform distributed computation meeting real-time requirements.

Our approach is one that minimizes the impact on the assurance and functionality of the existing B3 system base which we are extending. The real-time scheduling microkernel modifications enhance the assurance of the system by increasing modularity and defining security requirements for the use of alternative real-time scheduling algorithms. Because of the layered, modular, object-oriented structure of the TMach servers, the addition of Triad distribution functionality does not impact the existing TMach functionality of the server. The real-time changes to the servers—supporting migrating thread RPC—has very little effect on the existing implementation; the only difference being the initialization code that sets up activations for migrating threads. Indeed, a server can support both styles of RPC—migrating or non-migrating—by setting up both activations and standard threads in such a way that the same RPC operation is executed regardless of which style of RPC the client uses.

Thus, we have started with an existing extensible B3 MLS trusted system base, and established that real-time and distribution extensions can be made in a way that is consistent with the B3 level of assurance. Furthermore, we have identified that there is a small and manageable set of security requirements for the new real-time and distribution functionality—largely restricted to the distributed propagation of security attributes, and the identification of covert channels in real-time scheduling. We have analyzed the tradeoffs between the three areas, and determined the use of system-build techniques to manage these tradeoffs, both in the area of real-time (reducing the threat from scheduling covert channels) and distribution (replicating objects to be local to clients, to avoid incurring network communication overhead in real-time computation). Finally, we have determined that the Triad extensions of TMach are not only consistent with the existing subject/object definitions and policy of TMach, but also supportive of the concepts of operation and programming abstraction identified in the previous study phase of the project.

Therefore, we are confident that we are developing a prototype system that combines trusted MLS services with distributed real-time computation in a manner which supports emerging requirements for modern sophisticated real-time systems.

References

[1] *Trusted Mach System Architecture*, TIS TMach Edoc-0001-93B, Trusted Information Systems, Inc., 24 May 1993.

[2] Greenberg, Ira, et al., *The Multilevel Secure Real-Time Distributed Operating System Study*, RL-TR-93-101, Rome Laboratory, May 1993.

[3] Northcutt, J. Duane, et al, *Decentralized Computing Technology for Fault-Tolerant, Survivable C^3I Systems, Functional Description*, 1 December 1988.

[4] Benzel, T.C. Vickers, et al, *The Role of System Build in Trusted Embedded Systems*, Proceedings of the 13th National Computer Security Conference, Volume I, October 1990.

[5] Accatta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M., *Mach: A New Kernel Foundation for UNIX*, Proceedings of USENIX, July 1986.

[6] Floyd, Richard et al., *Future Directions for Replication in Cronus*, BBN Systems and Technologies Corporation, 16 April 1990.

[7] Jensen, E. Douglas, *A Timeliness Model For Scaleable Real-Time Computer Systems*, Transactions of DECUS, Fall 1992.

[8] Golub, D., *Adding Real-Time Scheduling to the Mach Kernel*, 1993, unpublished.

[9] Ford, Bryan, LePreau, Jay, *Evolving Mach 3.0 to a Migrating Thread Model*, Proceedings of USENIX Technical Conference, 17 January 1994.

[10] Burke, Condict, Mitchell, Reynolds, Watkins, Willcox, *RPC Design for Real-Time Mach*, Open Software Foundation/Research Institute, 12 April 1994.

[11] Orman, Hilarie, et al., *A Fast and General Implementation of Mach IPC in a Network*, Proceedings of USENIX Mach III Symposium, 19 April 1993.

AQ

22 JANUARY 1998

POC: ROME LAB
COMPUTER SYSTEMS BRANCH
ROME, NY 13441-3625